# Herramienta de Diagnóstico Fitopatológico Cuantitativo

Este proyecto es una aplicación web diseñada para el análisis cuantitativo de enfermedades foliares mediante el procesamiento digital de imágenes. La herramienta permite a los usuarios cargar imágenes que contienen una o varias hojas y realiza un análisis para detectar cada hoja individualmente, cuantificar la severidad del daño (área afectada) y contar el número de lesiones.

## 📋 Características Principales

* **Carga de Lotes de Imágenes:** Permite al usuario seleccionar múltiples archivos de imagen para un análisis eficiente.
* **Detección de Hojas Individuales:** Utiliza algoritmos de visión por computadora (detección de contornos) para identificar y separar múltiples hojas dentro de una misma fotografía.
* **Análisis Cuantitativo por Hoja:** Para cada hoja detectada, calcula:
  + **Porcentaje de Área Afectada:** Mide la severidad del daño.
  + **Conteo de Lesiones:** Cuantifica el número de focos de infección.
* **Visualización de Resultados:**
  + **Resumen Epidemiológico:** Muestra la incidencia y la severidad promedio del lote analizado.
  + **Panel Gráfico:** Gráficos de barras y de dispersión para una interpretación visual de los datos.
  + **Tabla de Datos Detallada:** Presenta los resultados numéricos de cada hoja individual.
* **Feedback Visual del Análisis:** Muestra una imagen procesada para cada hoja con las áreas de daño resaltadas con un filtro rojo, permitiendo al usuario validar el análisis del algoritmo.
* **Exportación de Datos:** Permite descargar un informe completo en formato **CSV** (compatible con Excel) con todos los datos obtenidos.
* **Registro Automático:** Opción para guardar un historial de todos los análisis en un archivo CSV en una carpeta local seleccionada por el usuario.

## 🏗️ Arquitectura de la Aplicación

La aplicación sigue una arquitectura moderna de **cliente-servidor**, lo que la hace robusta y escalable.

* **Backend (Servidor):** Desarrollado en **Python** con el framework **FastAPI**. Se encarga de toda la lógica pesada de procesamiento de imágenes utilizando la librería **OpenCV**. Es el "cerebro" de la aplicación.
* **Frontend (Cliente):** Una interfaz de usuario web construida con **HTML, CSS y JavaScript puro**. Se ejecuta en el navegador del usuario y su única función es proporcionar una forma amigable de interactuar con el backend. Es la "cara" de la aplicación.

[Imagen de a client-server architecture diagram](https://encrypted-tbn0.gstatic.com/licensed-image?q=tbn:ANd9GcQBDoZG161_7KJF1cjX81SwbeO_3qvegHbafrbGNgLBDylNteZBCX3qj4REXD_aERZlDBZ2NYwFMDPFyHrf_QPLa6pBZkJGLlxuC8wznFFsA8yTQMc)

## 🚀 Cómo Ejecutar el Proyecto

Para poner en marcha la aplicación en un entorno local, sigue estos pasos.

### **Requisitos Previos**

* Tener **Python** (versión 3.8 o superior) instalado.
* Tener **pip** (el gestor de paquetes de Python) actualizado.

### **Paso 1: Instalación de Dependencias**

Abre una terminal (cmd, PowerShell, etc.), navega hasta la carpeta principal del proyecto y ejecuta el siguiente comando. Este comando leerá el archivo requirements.txt e instalará todas las librerías necesarias.

Bash

pip install -r requirements.txt

### **Paso 2: Iniciar el Servidor de Python (Backend)**

En la misma terminal, ejecuta el siguiente comando para arrancar el servidor.

Bash

python -m uvicorn main:app --reload

Este comando le dice a uvicorn que ejecute la aplicación app que se encuentra en el archivo main.py. La opción --reload hace que el servidor se reinicie automáticamente si haces cambios en el código.

Si todo va bien, verás un mensaje que dice: Uvicorn running on http://127.0.0.1:8000. **¡No cierres esta terminal!**

### **Paso 3: Abrir la Aplicación Web (Frontend)**

Ve a la carpeta fitopatologia-frontend y haz doble clic en el archivo index.html. Esto abrirá la interfaz de usuario en tu navegador web predeterminado. La aplicación ya está lista para usarse.

## 🔬 Explicación Detallada del Código

Aquí se desglosan las partes más importantes del código y su funcionamiento.

### **Backend: main.py**

Este archivo es el núcleo de la aplicación. Su trabajo se divide en dos etapas principales.

#### **Función detect\_leaves(img\_bgr):**

Esta función es la **Etapa 1**. Su objetivo es encontrar todas las hojas en la imagen original.

* **Líneas de Código Clave:**  
  Python  
  # Convertir a escala de grises y aplicar desenfoque para reducir ruido  
  img\_gray = cv2.cvtColor(img\_bgr, cv2.COLOR\_BGR2GRAY)  
  img\_blur = cv2.GaussianBlur(img\_gray, (7, 7), 0)  
    
  # Separar los objetos del fondo usando un umbral adaptativo (Otsu)  
  \_, thresh = cv2.threshold(img\_blur, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)  
    
  # Limpiar la imagen binaria para eliminar ruido y rellenar huecos  
  kernel = np.ones((5, 5), np.uint8)  
  thresh\_cleaned = cv2.morphologyEx(thresh, cv2.MORPH\_CLOSE, kernel, iterations=2)  
    
  # Encontrar los contornos (bordes) de los objetos restantes  
  contours, \_ = cv2.findContours(thresh\_cleaned, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)
* **¿Por qué y Para qué?** Este proceso, aunque no es una IA, es un método de visión por computadora robusto para la **segmentación de objetos**. Transforma la imagen en blanco y negro para encontrar formas bien definidas (los contornos de las hojas) y las devuelve como una lista de recuadros que serán analizados individualmente.

#### **Función analyze\_single\_leaf(leaf\_image):**

Esta función es la **Etapa 2**. Recibe una imagen ya recortada de una sola hoja y la analiza.

* **Líneas de Código Clave:**  
  Python  
  # Convertir al espacio de color HLS para un análisis de color robusto  
  img\_hls = cv2.cvtColor(leaf\_image, cv2.COLOR\_BGR2HLS)  
    
  # Crear máscaras para tejido sano (verde) y dañado (amarillo/marrón)  
  healthy\_mask = cv2.inRange(img\_hls, (39, 51, 25), (89, 255, 255))  
  damaged\_mask = cv2.inRange(img\_hls, (10, 51, 25), (36, 255, 255))  
    
  # Contar los píxeles de cada máscara para calcular el área  
  damaged\_pixels = cv2.countNonZero(damaged\_mask)  
  healthy\_pixels = cv2.countNonZero(healthy\_mask)  
  area\_damage = (damaged\_pixels / (damaged\_pixels + healthy\_pixels)) \* 100  
    
  # Contar los grupos de píxeles dañados para obtener el número de lesiones  
  num\_labels, \_, \_, \_ = cv2.connectedComponentsWithStats(damaged\_mask, 4, cv2.CV\_32S)  
  lesion\_count = num\_labels - 1
* **¿Por qué y Para qué?** Esta es la lógica de cuantificación. Al usar el espacio de color HLS, el análisis es menos sensible a sombras. Las máscaras permiten contar píxeles de forma precisa para obtener el **porcentaje de severidad**. El algoritmo connectedComponentsWithStats es la forma estándar en visión por computadora para contar objetos discretos en una máscara, en este caso, las lesiones.

### **Frontend: script.js**

Este archivo gestiona toda la interactividad del usuario.

#### **Función handleFiles(files):**

Se activa cuando el usuario selecciona imágenes.

* **¿Cómo funciona?** Utiliza la API FileReader de JavaScript. Para cada archivo de imagen, lee su contenido y lo convierte en una cadena de texto (Data URL) que puede ser usada como la fuente (src) de un elemento <img> de HTML. Esto permite mostrar las previsualizaciones sin necesidad de subir los archivos al servidor todavía.

#### **Función analyzeButton.addEventListener('click', ...):**

Es el corazón de la interacción. Se activa al hacer clic en "Ejecutar Análisis".

* **¿Cómo funciona?**
  1. Inicia un bucle que recorre cada imagen seleccionada por el usuario.
  2. Para cada imagen, crea un objeto FormData, que es la forma estándar de empaquetar archivos para enviarlos a un servidor.
  3. Usa la función fetch de JavaScript para enviar el FormData al endpoint /analizar-imagen-compleja/ del backend de Python de forma asíncrona (la página no se congela).
  4. Espera la respuesta del servidor, que llega en formato JSON.
  5. Decodifica el JSON y, como el backend devuelve un resultado por cada hoja detectada, aplana esta lista para que cada hoja sea una entrada individual en la tabla de resultados.
  6. Una vez que todas las imágenes han sido procesadas, llama a las funciones displayDataTable, displaySummary y createCharts para actualizar la interfaz con los resultados finales.